FEM Integration with Quadrature on the GPU

Matthew Knepley

Computation Institute
University of Chicago

Department of Molecular Biology and Physiology
Rush University Medical Center

GPU-SMP 2012
Shenzhen, China June 1–4, 2012

Collaborators

Andy R. Terrel

- Andreas Klöckner
- Jed Brown
- Robert Kirby

Outline

Main Point

To be widely accepted,

GPU computing must be transparent to the user,

and reuse existing infrastructure.

Main Point

To be widely accepted,

GPU computing must be
transparent to the user,

and reuse existing
infrastructure.

Main Point

To be widely accepted,

GPU computing must be transparent to the user,

and reuse existing infrastructure.

Lessons from Clusters and MPPs

Failure

- Parallelizing Compilers
- Automatic program decomposition

Success

- MPI (Library Approach)
- PETSc (Parallel Linear Algebra)
- User provides only the mathematical description

Lessons from Clusters and MPPs

Failure

- Parallelizing Compilers
- Automatic program decomposition

Success

- MPI (Library Approach)
- PETSc (Parallel Linear Algebra)
- User provides only the mathematical description

Outline

What is PETSc?

A freely available and supported research code for the parallel solution of nonlinear algebraic equations

Free

- Download from http://www.petsc.org
- Free for everyone, including industrial users

Supported

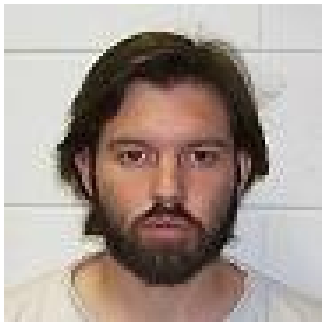
- Hyperlinked manual, examples, and manual pages for all routines
- Hundreds of tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov

Usable from C, C++, Fortran 77/90, Matlab, Julia, and Python

What is PETSc?

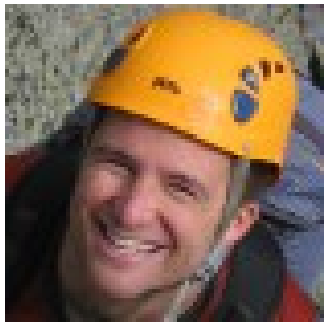
- Portable to any parallel system supporting MPI, including:
  - Tightly coupled systems
    - Cray XT6, BG/Q, NVIDIA Fermi, K Computer
  - Loosely coupled systems, such as networks of workstations
    - IBM, Mac, iPad/iPhone, PCs running Linux or Windows
- PETSc History
  - Begun September 1991
  - Over 60,000 downloads since 1995 (version 2)
  - Currently 400 per month
- PETSc Funding and Support
  - Department of Energy
    - ECP, PSAAPIII, AMR, BES, SciDAC, MICS
  - National Science Foundation
    - CSSI, SI2, CIG, CISE
  - Intel Parallel Computing Center

The PETSc Team


Matt Knepley


Barry Smith


Satish Balay


Jed Brown

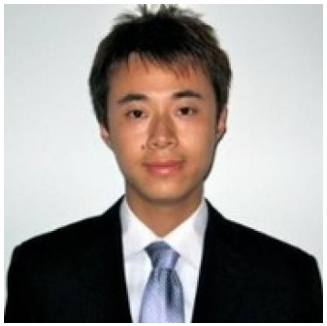

Hong Zhang

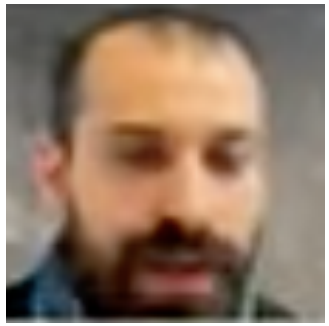

Lisandro Dalcin


Stefano Zampini


Mark Adams


Toby Isaac


Hong Zhang


Pierre Jolivet


Junchao Zhang

Who Uses PETSc?

Computational Scientists

- Earth Science
  - PyLith (CIG)
  - Underworld (Monash)
  - Salvus (ETHZ)
  - TerraFERMA (LDEO, Columbia, Oxford)
- Multiphysics
  - MOOSE
  - GRINS
- Subsurface Flow and Porous Media
  - PFLOTRAN (DOE)
  - STOMP (DOE)

Who Uses PETSc?

Computational Scientists

- CFD
  - IBAMR
  - Fluidity
  - OpenFVM
- Fusion
  - XGC
  - BOUT++
  - NIMROD
  - $M3D - C^1$

Who Uses PETSc?

Algorithm Developers

- Iterative methods
  - Deflated GMRES
  - LGMRES
  - QCG
  - SpecEst
- Preconditioning researchers
  - FETI-DP (Klawonn and Rheinbach)
  - STRUMPACK (Ghysels and Li)
  - HPDDM (Jolivet and Nataf)
  - ParPre (Eijkhout)

Who Uses PETSc?

Algorithm Developers

- Discretization
  - Firedrake
  - FEniCS
  - libMesh
  - Deal II
  - PETSc-FEM
  - OOFEM
  - PetRBF
- Outer Loop Solvers
  - Eigensolvers (SLEPc)
  - Optimization (PERMON)

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Interface Questions

How should the user interact with manycore systems?

Through computational libraries

How should the user interact with the library?

Strong, data structure-neutral API (Smith and Gropp, 1996)

How should the library interact with manycore systems?

- Existing library APIs
- Code generation (CUDA, OpenCL, PyCUDA)
- Custom multi-language extensions

Outline

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

PETSc-GPU

PETSc now has support for Krylov solves on the GPU

- `-with-cuda=1 -with-cusp=1 -with-thrust=1`
  - Also possibly `-with-precision=single`
- New classes `VECCUDA` and `MATAIJCUDA`
  - Just change type on command line, `-vec_type veccuda`
- Uses Thrust and Cusp libraries from Nvidia guys
- Does not communicate vectors during solve

Example

Driven Cavity Velocity-Vorticity with Multigrid

```
ex50 -da_vec_type seqcusp
  -da_mat_type aijcusp -mat_no_inode # Setup types
  -da_grid_x 100 -da_grid_y 100      # Set grid size
  -pc_type none -pc_mg_levels 1      # Setup solver
  -preload off -cuda_synchronize     # Setup run
  -log_summary
```

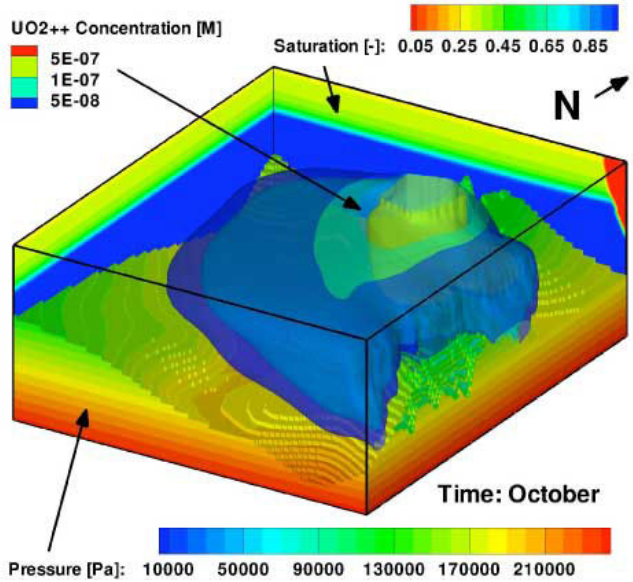
Example

PFLOTRAN

Flow Solver

$32 \times 32 \times 32$ grid

Routine	Time (s)	MFlops	MFlops/s
CPU			
KSPSolve	8.3167	4370	526
MatMult	1.5031	769	512
GPU			
KSPSolve	1.6382	4500	2745
MatMult	0.3554	830	2337



P. Lichtner, G. Hammond, R. Mills, B. Phillip

Outline

Form Decomposition

Element integrals are decomposed into analytic and geometric parts:

$$\int_{\mathcal{T}} \nabla\phi_i(\mathbf{x}) \cdot \nabla\phi_j(\mathbf{x}) d\mathbf{x} \tag{1}$$

$$= \int_{\mathcal{T}} \frac{\partial\phi_i(\mathbf{x})}{\partial x_\alpha} \frac{\partial\phi_j(\mathbf{x})}{\partial x_\alpha} d\mathbf{x} \tag{2}$$

$$= \int_{\mathcal{T}_{\text{ref}}} \frac{\partial\xi_\beta}{\partial x_\alpha} \frac{\partial\phi_i(\xi)}{\partial\xi_\beta} \frac{\partial\xi_\gamma}{\partial x_\alpha} \frac{\partial\phi_j(\xi)}{\partial\xi_\gamma} |J| d\mathbf{x} \tag{3}$$

$$= \frac{\partial\xi_\beta}{\partial x_\alpha} \frac{\partial\xi_\gamma}{\partial x_\alpha} |J| \int_{\mathcal{T}_{\text{ref}}} \frac{\partial\phi_i(\xi)}{\partial\xi_\beta} \frac{\partial\phi_j(\xi)}{\partial\xi_\gamma} d\mathbf{x} \tag{4}$$

$$= G^{\beta\gamma}(\mathcal{T}) K^{ij}_{\beta\gamma} \tag{5}$$

Coefficients are also put into the geometric part.

Tensor Product Formulation

FEniCS based code achieves

90 GF/s on 3D P_1 Laplacian

100 GF/s on 2D P_1 Elasticity

- Relies on analytic integration
- Dot products are workhorse
- Crossover point with quadrature with multiple fields

Finite Element Integration on GPUs, ACM TOMS, Andy R. Terrel and Matthew G. Knepley

Why Quadrature?

Quadrature can handle

- many fields (linearization)
- non-affine elements (Argyris)
- non-affine mappings (isoparametric)
- functions not in the FEM space

Optimizations for Quadrature Representations of Finite Element Tensors through Automated Code Generation, ACM TOMS, Kristian B. Ølgaard and Garth N. Wells

Jed Brown's Model

We consider weak forms dependent only on fields and gradients,

$$\int_\Omega \phi \cdot f_0(u, \nabla u) + \nabla\phi : \mathbf{f}_1(u, \nabla u) = 0. \tag{6}$$

Discretizing we have

$$\sum_e \mathcal{E}_e^T \left[B^T W^q f_0(u^q, \nabla u^q) + \sum_k D_k^T W^q \mathbf{f}_1^k(u^q, \nabla u^q) \right] = 0 \tag{7}$$

f_n	pointwise physics functions
u_q	field at a quad point
W^q	diagonal matrix of quad weights
B,D	basis function matrices which reduce over quad points
$\mathcal{E}$	assembly operator

Physics code

$$\nabla \phi_i \cdot \nabla u$$

Physics code

$$\nabla \phi_i \cdot \nabla u$$

```
__device__ vecType f1(realType u[], vecType gradU[], int comp) {
  return gradU[comp];
}
```

Physics code

$$\nabla \phi_i \cdot (\nabla u + \nabla u^T)$$

Physics code

$$\nabla \phi_i \cdot (\nabla u + \nabla u^T)$$

```
__device__ vecType f1(realType u[], vecType gradU[], int comp) {
  vecType f1;

  switch(comp) {
  case 0:
    f1.x = 0.5*(gradU[0].x + gradU[0].x);
    f1.y = 0.5*(gradU[0].y + gradU[1].x);
    break;
  case 1:
    f1.x = 0.5*(gradU[1].x + gradU[0].y);
    f1.y = 0.5*(gradU[1].y + gradU[1].y);
  }
  return f1;
}
```

Physics code

$$\nabla \phi_i \cdot \nabla u + \phi_i k^2 u$$

Physics code

$$\nabla \phi_i \cdot \nabla u + \phi_i k^2 u$$

```
__device__ vecType f1(realType u[], vecType gradU[], int comp) {
  return gradU[comp];
}

__device__ realType f0(realType u[], vecType gradU[], int comp) {
  return k*k*u[0];
}
```

Physics code

$$\nabla \phi_i \cdot \nabla \mathbf{u} - (\nabla \cdot \phi) p$$

Physics code

$$\nabla\phi_i \cdot \nabla\mathbf{u} - (\nabla \cdot \phi)p$$

```
void f1(PetscScalar u[], const PetscScalar gradU[], PetscScalar f1[]) {
  const PetscInt dim   = SPATIAL_DIM_0;
  const PetscInt Ncomp = NUM_BASIS_COMPONENTS_0;
  PetscInt       comp, d;

  for (comp = 0; comp < Ncomp; ++comp) {
    for (d = 0; d < dim; ++d) {
      f1[comp*dim+d] = gradU[comp*dim+d];
    }
    f1[comp*dim+comp] -= u[Ncomp];
  }
}
```

Physics code

$$\nabla\phi_i \cdot \nu_0 e^{-\beta T}\nabla\mathbf{u} - (\nabla\cdot\phi)p$$

Physics code

$$\nabla\phi_i \cdot \nu_0 e^{-\beta T} \nabla \mathbf{u} - (\nabla \cdot \phi) p$$

```
void f1(PetscScalar u[], const PetscScalar gradU[], PetscScalar f1[]) {
  const PetscInt dim   = SPATIAL_DIM_0;
  const PetscInt Ncomp = NUM_BASIS_COMPONENTS_0;
  PetscInt       comp, d;

  for (comp = 0; comp < Ncomp; ++comp) {
    for (d = 0; d < dim; ++d) {
      f1[comp*dim+d] = nu_0*exp(-beta*u[Ncomp+1])*gradU[comp*dim+d];
    }
    f1[comp*dim+comp] -= u[Ncomp];
  }
}
```

Why Not Quadrature?

Vectorization is a Problem

Strategy	Problem

Why Not Quadrature?

Vectorization is a Problem

Strategy	**Problem**
Vectorize over Quad Points	Reduction needed to compute Basis Coefficients

Why Not Quadrature?

Vectorization is a Problem

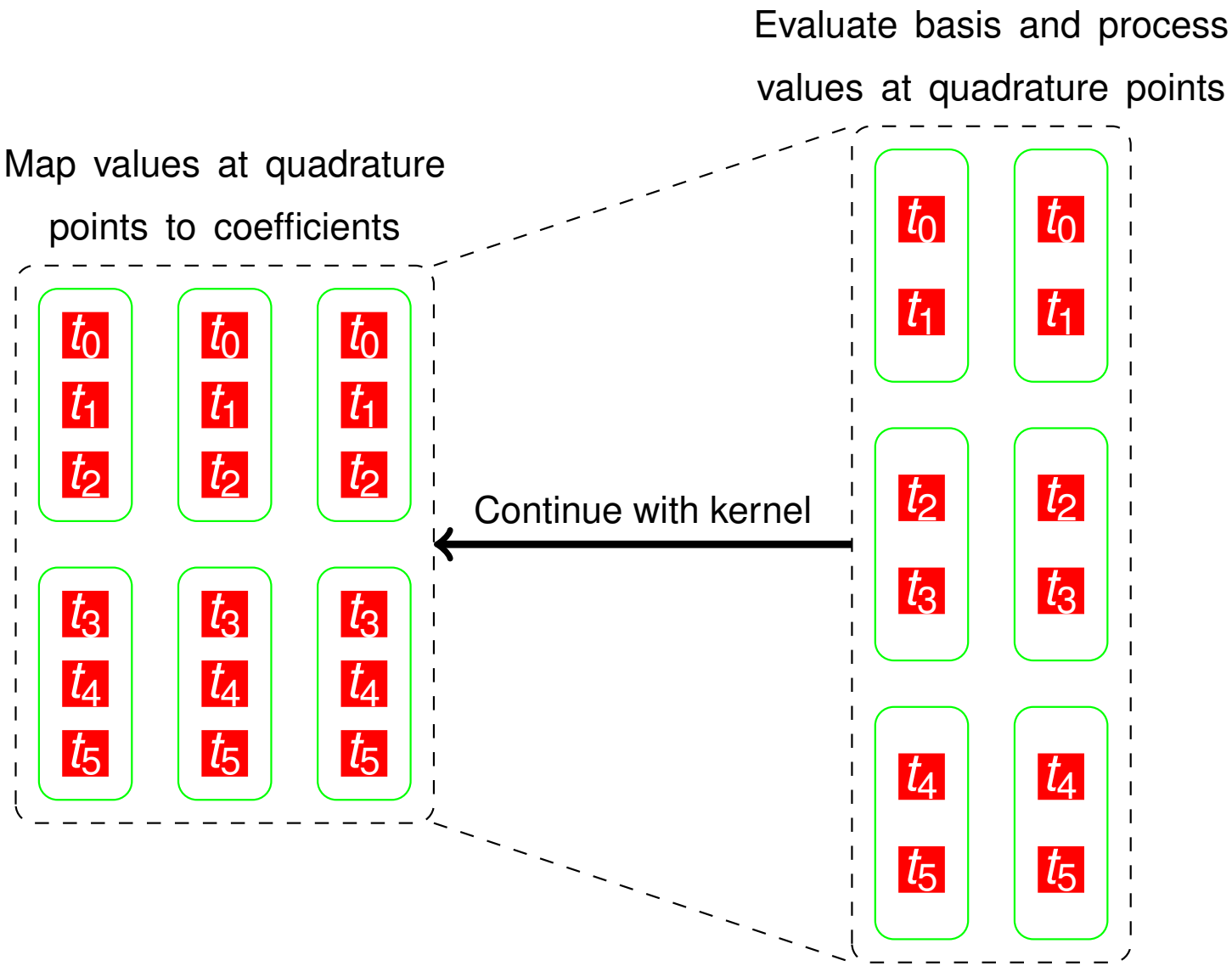
Strategy	**Problem**
Vectorize over Quad Points	Reduction needed to compute Basis Coefficients
Vectorize over Basis Coef for each Quad Point	Too many passes through global memory

Why Not Quadrature?

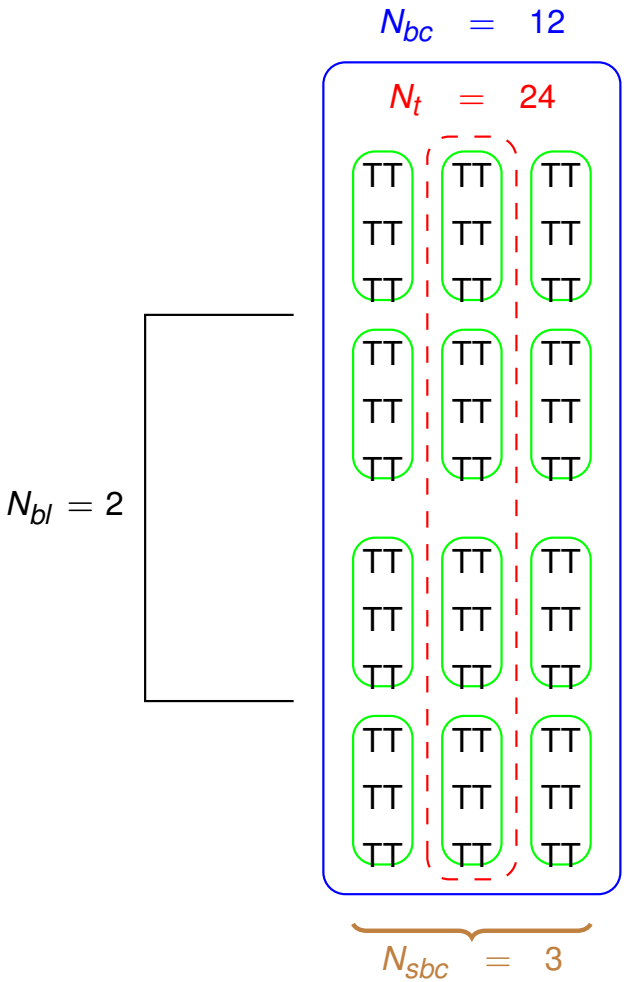
Vectorization is a Problem

Strategy	**Problem**
Vectorize over Quad Points	Reduction needed to compute Basis Coefficients
Vectorize over Basis Coef for each Quad Point	Too many passes through global memory
Vectorize over Basis Coef and Quad Points	Some threads idle when sizes are different

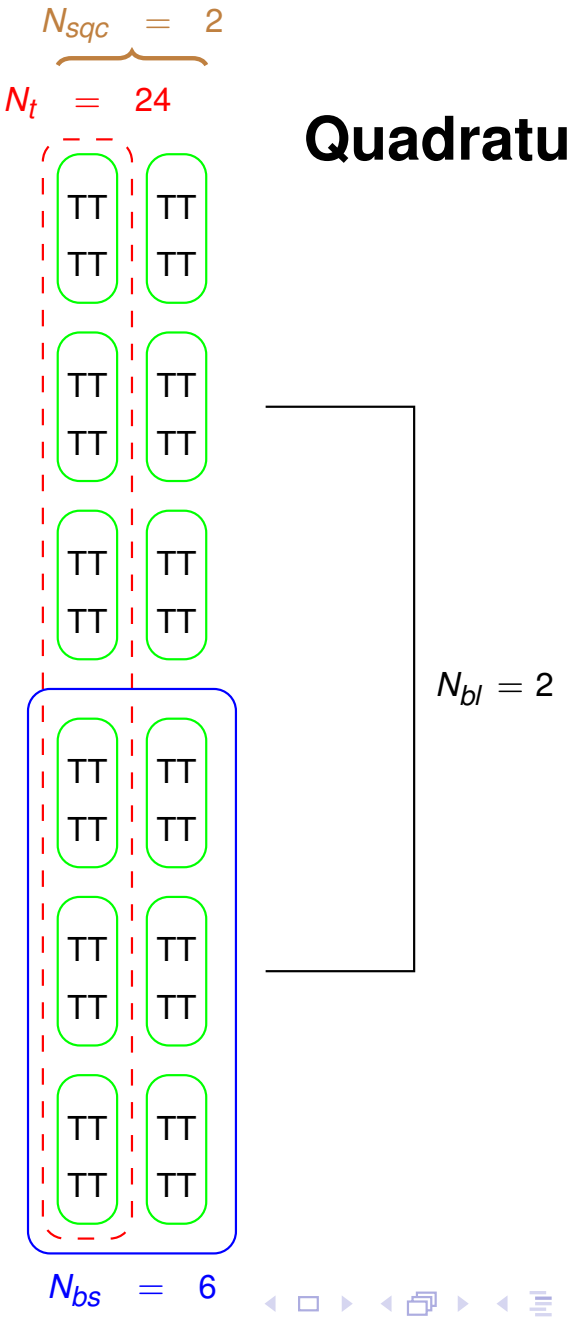
Thread Transposition



Basis Phase



Quadrature Phase



PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		**Automation**		**Solver Input**
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- Loops are done in batches
- Remainder cells are integrated on the CPU
- PETSc ex52 is a single-field example

PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- Loops are done in batches
- Remainder cells are integrated on the CPU
- PETSc ex52 is a single-field example

PETSc Multiphysics

Each block of the Jacobian is evaluated separately:

- Reuse single-field code
- Vectorize over cells, rather than fields
- Retain sparsity of the Jacobian

Solver integration is seamless:

- Nested Block preconditioners from the command line
- Segregated KKT MG smoothers from the command line
- Fully composable with AMG, LU, Schur complement, etc.

PETSc ex62 solves the Stokes problem,
and ex31 adds temperature

PETSc Multiphysics

Each block of the Jacobian is evaluated separately:

- Reuse single-field code
- Vectorize over cells, rather than fields
- Retain sparsity of the Jacobian

Solver integration is seamless:

- Nested Block preconditioners from the command line
- Segregated KKT MG smoothers from the command line
- Fully composable with AMG, LU, Schur complement, etc.

PETSc ex62 solves the Stokes problem,
and ex31 adds temperature

PETSc Multiphysics

Each block of the Jacobian is evaluated separately:

- Reuse single-field code
- Vectorize over cells, rather than fields
- Retain sparsity of the Jacobian

Solver integration is seamless:

- Nested Block preconditioners from the command line
- Segregated KKT MG smoothers from the command line
- Fully composable with AMG, LU, Schur complement, etc.

PETSc ex62 solves the Stokes problem,
and ex31 adds temperature

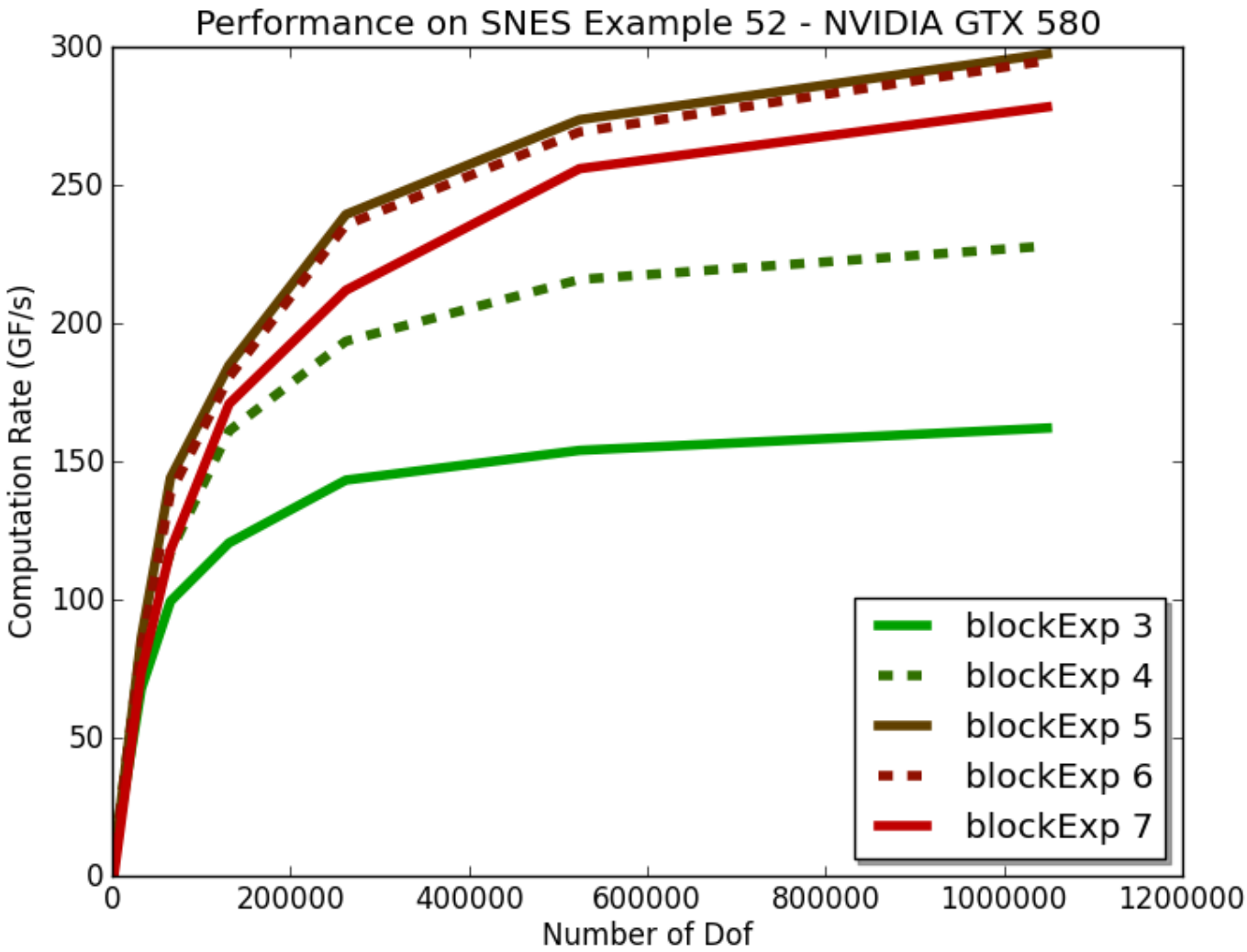
Performance Expectations

Element Integration

FEM Integration, at the element level, is also limited by memory bandwidth, rather than by peak flop rate.

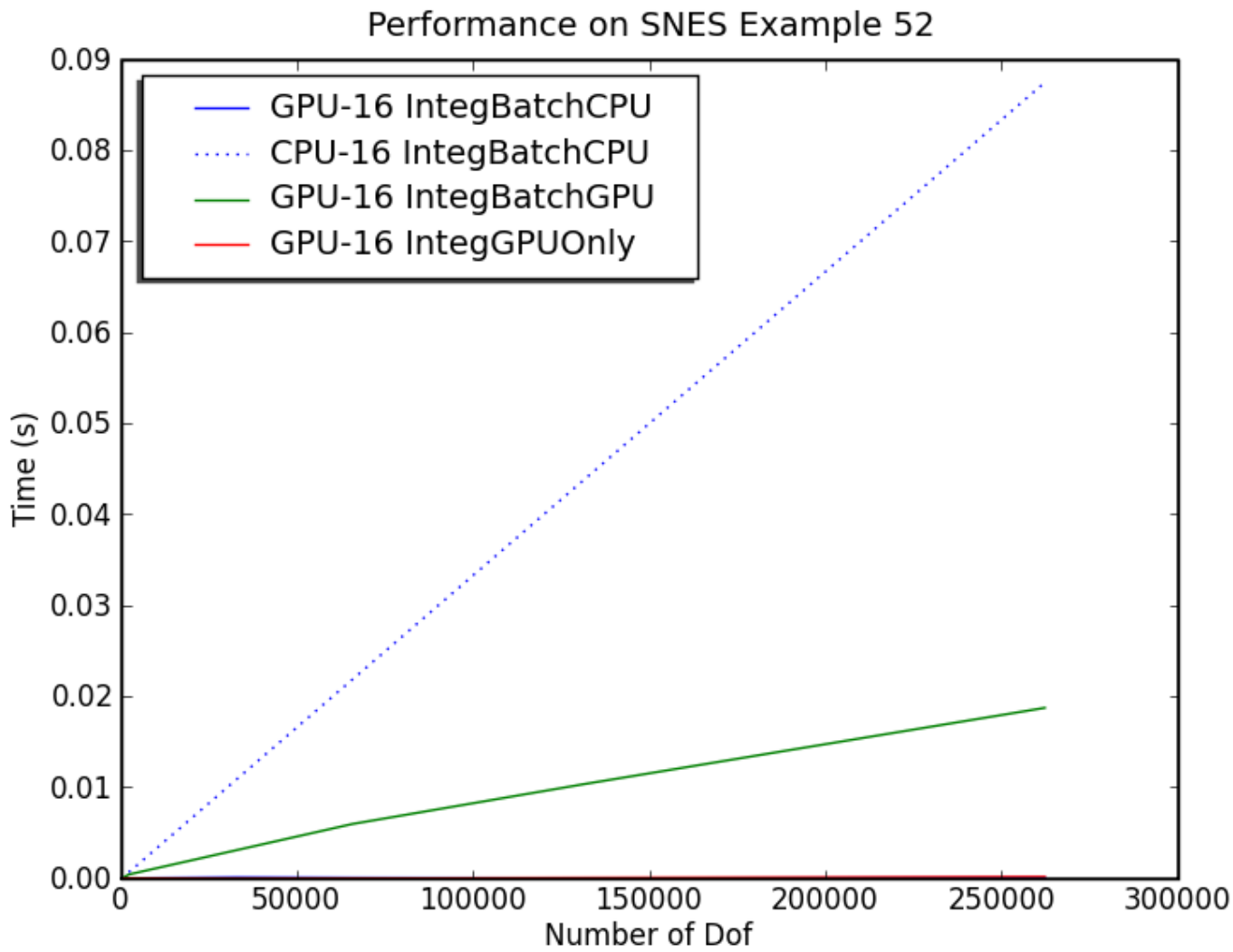
- We expect bandwidth ratio speedup (3x–6x for most systems)
- Input for FEM is a vector of coefficients (auxiliary fields)
- Output is a vector of coefficients for the residual

2D P_1 Laplacian Performance



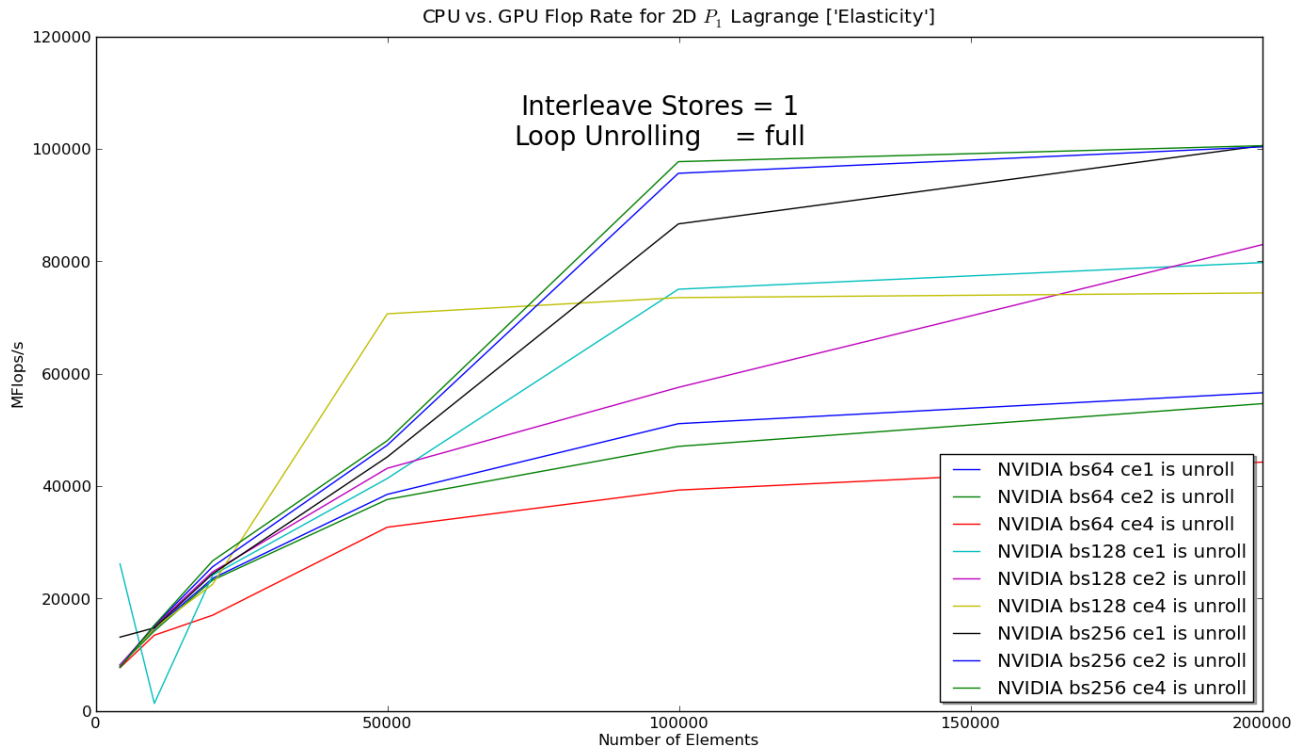
Reaches 100 GF/s by 100K elements

2D P_1 Laplacian Performance



Linear scaling for both GPU and CPU integration

2D P_1 Rate-of-Strain Performance



Reaches 100 GF/s by 100K elements

Outline

Competing Models

How should kernels be integrated into libraries?

CUDA+Code Generation

- Explicit vectorization
- Can inspect/optimize code
- Errors easily localized
- Can use high-level reasoning for optimization (FErari)
- Kernel fusion is easy

TBB+C++ Templates

- Implicit vectorization
- Generated code is hidden
- Notoriously difficult debugging
- Low-level compiler-type optimization
- Kernel fusion is really hard

Competing Models

How should kernels be integrated into libraries?

CUDA+Code Generation

- Explicit vectorization
- Can inspect/optimize code
- Errors easily localized
- Can use high-level reasoning for optimization (FErari)
- Kernel fusion is easy

TBB+C++ Templates

- Implicit vectorization
- Generated code is hidden
- Notoriously difficult debugging
- Low-level compiler-type optimization
- Kernel fusion is really hard

Competing Models

How should kernels be integrated into libraries?

CUDA+Code Generation

- Explicit vectorization
- Can inspect/optimize code
- Errors easily localized
- Can use high-level reasoning for optimization (FErari)
- Kernel fusion is easy

TBB+C++ Templates

- Implicit vectorization
- Generated code is hidden
- Notoriously difficult debugging
- Low-level compiler-type optimization
- Kernel fusion is really hard

Competing Models

How should kernels be integrated into libraries?

CUDA+Code Generation

- Explicit vectorization
- Can inspect/optimize code
- Errors easily localized
- Can use high-level reasoning for optimization (FErari)
- Kernel fusion is easy

TBB+C++ Templates

- Implicit vectorization
- Generated code is hidden
- Notoriously difficult debugging
- Low-level compiler-type optimization
- Kernel fusion is really hard

Competing Models

How should kernels be integrated into libraries?

CUDA+Code Generation

- Explicit vectorization
- Can inspect/optimize code
- Errors easily localized
- Can use high-level reasoning for optimization (FErari)
- Kernel fusion is easy

TBB+C++ Templates

- Implicit vectorization
- Generated code is hidden
- Notoriously difficult debugging
- Low-level compiler-type optimization
- Kernel fusion is really hard